

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Xml;
using System.Xml.Linq;
using System.Xml.XPath;
using System.IO;

namespace Gameplay
{
    /*This is the song class - this is the primary source for holding the note information once
    it has been pulled from the MusicXML document*/
    public class TSong
    {
        public TBaseNote[] songNotes;           //The game ntoes- inc chords
        public TBaseNote[] allXMLNotes;         //The xml notes - pre chord sorting (could be a list of TBaseNote too but just to help differentiate
        public XDocument xml;

        public int smallestDivision;
        public int numberOfBars;
        public int origNumberOfNotes;           //this is the number of notes before being sorted into chords
        //-----
        public int numberOfNotes;               //this is the number of notes after chord groupings (all notes in a chord count as 1 notes as are placed into a class)
        public int totalBeats;                  //the total number of beats within the song

        /*This is the constructor for the TSong where the MusicXML document source is held and the
        * song information is populated firstly into a list holding all notes as if they were single notes
        * then once this list is used to create another list where the chords are built appropriately*/
        public TSong(String xmlDocName)
        {
            xml = XDocument.Load(xmlDocName);    //store the source location

            calcNumberOfUnsortedNotes();         //calc the number of all notes (pre chord)
            calcNumberOfSortedNotes();           //calc number of note positions (this means all notes in a chord count as 1)

            allXMLNotes = new TBaseNote[origNumberOfNotes]; //create an array with locations for all notes counted as singlt notes
            songNotes = new TBaseNote[numberOfNotes];       //create an array with locations only for chords

            findSmallestDivision();               //calculate the smallest division for the song
            calcNumberOfBars();                   //calculate the number of bars within the piece

            fillOrigSongNotes();                  //fill the original song array- all notes as single ntoes
            fillChordInclusiveNotes();            //usw the original array and create an array that holds all the song with chords correctly recorded

            recomputeChordArrayTiming();          //because the chords can have 3 notes and all notes need to start on the same time spot for checking.

            allXMLNotes = null;                   //set the original notes array to null to free the memory as it is never used again
        }

        //*****

        /*This is the insert chord method that accepts the start location of a known chord, creates a new note
        * from the starting point and has the subsequent note details from the other notes added into the
        * new chord note. Then this is added to the note array that contains the proper notes with chords too*/
        public int insertChord(int unsortedNoteLoc, int sortedNoteLoc)
        {
            int newUnsortedNoteLoc = unsortedNoteLoc;    //keep track of where you are looking in the non-chord inclusive array

            bool inChord = true;                          //flag to state wheather you are still in the chord

            int chordTime = allXMLNotes[newUnsortedNoteLoc].theTime;    //time of first note in chord... will be changed later anyway
            int chordDuration = allXMLNotes[newUnsortedNoteLoc].theDuration; //Duration is set to that of the first note though it doesnt matter because the times need to be re-calculated later

```

```

TBaseNote currentChord = allXMLNotes[newUnsortedNoteLoc]; //assign start guy- if problems then could be here for pointer probs not doing copy properly

newUnsortedNoteLoc++; //we can incrememnt without error checking because we know that it is in a chord so there must be note to follow

TNoteDetail tempDetail;
String theNote;
String theString;
String theFret;

//while the in chord is set to true- (determined by the chord tags within the song notes
//(consecutive notes have the tag and means they are added on to the previous guy)
while (inChord)
{
    //for each TNoteDetail in the unsorted array (chord not built, also this will always be 1 Detail but foreach is safest way to check)
    foreach (TNoteDetail t in allXMLNotes[newUnsortedNoteLoc].myNoteDetails)
    {
        //pull the information/Detail from the note
        theNote = t.theNote;
        theString = t.theString;
        theFret = t.theFret;

        //create a new Detail from the above information
        tempDetail = new TNoteDetail(theNote, theString, theFret);

        //add to the chord 'BaseNote'
        currentChord.addNoteDetail(tempDetail);
    }

    //if next note has chord tag continue...
    if ((newUnsortedNoteLoc + 1) < origNumberOfNotes) //also check if the last note in the piece of music
    {
        if (allXMLNotes[newUnsortedNoteLoc + 1].chord == false)
            inChord = false;
    }
    else
    {
        inChord = false;
    }

    //increment the note index where you are looking into (in the unsorted array)
    newUnsortedNoteLoc++;
}

songNotes[sortedNoteLoc] = currentChord; //add the newly built note with Details to make up a chord to the songNotes.

return newUnsortedNoteLoc; //return the location of where you stopped looking at in the unsorted array to carry on from
}

//*****

/*This is the function to build the SongNote data that include the Chords built - this means the
original song notes array is traversed, if a sing note it is added to the new array and if it is
a chord then it is sent to the build chord function where Detail of the consecutive notes with
the chord tags are bundled into 1 note with all the TDetails to make up a chord- following this
function the timings of all the notes are recomputed as they are put out of sync when a chord
is built as pre chord each Detail in the Chord is recorded as a note itself so its timing gets
mucked up with duration being added each time*/
private void fillChordInclusiveNotes()
{
    int unsortedNoteLoc = 0; //indexes for the two arrays
    int sortedNoteLoc = 0;

    bool lastNote = false; //last note flag

    while (unsortedNoteLoc < origNumberOfNotes) //for every note in the array (unsorted)
    {
        lastNote = false;

        if((unsortedNoteLoc + 1) >= origNumberOfNotes) // make sure you are not going to peek off the array- if so then signal last note
            lastNote = true;

        if (lastNote)
        {
            //add single note --- cant be a chord and cant be part of a chord because of the add chord function workings

```

```

        songNotes[sortedNoteLoc] = allXMLNotes[unsortedNoteLoc];
        unsortedNoteLoc++;
    }
    else //if not the last note
    {
        if (allXMLNotes[unsortedNoteLoc + 1].chord == true)
        {
            //calls the function to create and insert the chord that starts at the current location
            //returns the value of the next note to look at after adding the chord as this is multiple notes from the orig array
            unsortedNoteLoc = insertChord(unsortedNoteLoc, sortedNoteLoc);
        }
        else
        {
            //add the single note from the unsorted array into the final chord inclusive array
            songNotes[sortedNoteLoc] = allXMLNotes[unsortedNoteLoc];
            unsortedNoteLoc++;
        }
    }

    sortedNoteLoc++;
}

}

//*****

/*This function traverses the Song notes array that includes all notes correctly built and
 * calculates their correct times. This is done by starting at a beat count of 1 and
 * adding the note durations on to therefore determining the start point for the next note*/
public void recomputeChordArrayTiming()
{
    int beatCount = 1;
    int duration = 0;

    //for every note in the SongNote array
    for (int i = 0; i < numberOfNotes; i++)
    {
        //duration for all notes in the chord will be the same, pull offs are not recorded
        //in here and if they were they would be recognised and a tie to another chord
        duration = songNotes[i].theDuration;

        songNotes[i].theTime = beatCount; //set the new time value

        beatCount += duration; //compute the start point of the next note
    }

    totalBeats = beatCount; //store the total number of beats in the piece
}

//*****

/*This function is used to populate the original song note information array
 * it treats every note in the MusicXML document as a single note and adds it to
 * an array. this is then used as a source array to populate the actual array that
 * include chords to be held in a more suitable way so that the timings are able to
 * be set properly and the structure provides the ability to do chord detection comparison
 * analysis as well.*/
public void fillOrigSongNotes()
{
    //Setting up the default values- names
    int beatCount = 1;
    int currLoc = 0;
    String theFret = "0";
    String theString = "0";
    int theDuration = 0;
    String note = " ";

    bool chord = false;

    int origDuration = 0;
    int currBarDivision = 0;

    //state the Base note and NoteDetail class that will be used when adding note to the array
    TBaseNote tempBase;
    TNoteDetail tempDetail;

```

```

//create a list of infotmation that includes all data within the 'measure' (bar) tags from
//the xml document.
var query = from MusicFile in xml.Elements("score-partwise").Elements("part").Elements("measure")
            select MusicFile;

//for each measure in the list of data/query
foreach (var Measure in query)                                //loop through each measure
{
    //finding division size of current bar to calculate the duration size in relation to smallest division of the SONG
    if (Measure.Descendants("attributes").Any())                //if there is a division value (as if the same as previous bar it is not stated)
    {
        currBarDivision = Convert.ToInt16(Measure.Element("attributes").Element("divisions").Value);    //store division value into current divisino value
    }

    //loop through for each note in the measure
    foreach (var Note in Measure.Elements("note"))
    {
        chord = false;    //default the chord flag to false.

        if (Note.Descendants("pitch").Any())                    //making sure the pitch tag exists? this must check if there is a note
        {
            if (Note.Descendants("chord").Any())                //look for the chord tag- if it is present set the flag to true
                chord = true;

            origDuration = 0;                                    //default the duration

            note = Note.Element("pitch").Element("step").Value;    //get the current note

            if (Note.Element("pitch").Descendants("alter").Any())    //this is to make sure if there is an alter if takes it into account otherwise nothing extra happens
            {
                if (Note.Element("pitch").Element("alter").Value == "1")    //checking for sharps - if it doesnt have an alter it crashes so we need to put in a if exists?
                    note += "#";

                //testing purposes
                //MessageBox.Show("the note: " + note);
            }

            theFret = Note.Element("notations").Element("technical").Element("fret").Value;    //store the fret value

            theString = Note.Element("notations").Element("technical").Element("string").Value;    //store the string value

            origDuration = Convert.ToInt16(Note.Element("duration").Value);    //this is the same as the time code below so could be combined into 1

            theDuration = (smallestDivision / currBarDivision) * origDuration;    //calculate the duration of the note in realtion to the whole piece not just its current bar

            tempBase = new TBaseNote(beatCount, theDuration, chord);    //Create the base class- hold the time, duration and chord tag
            tempDetail = new TNoteDetail(note, theString, theFret);    //creat the deatail of the corosponding note- string fret and note name
            tempBase.addNoteDetail(tempDetail);    //add the detail to the base note class

            allXMLNotes[currLoc] = tempBase;    //add the complete note to the array

            beatCount += theDuration;    //add to the total beat count - used for next notes start location
            currLoc++;
        }
        else    //otherwise if a rest
        {
            int restDurationOrig = Convert.ToInt16(Note.Element("duration").Value);    //get the duration value

            //default the values for a rest
            note = "R";
            theFret = "-1";
            theString = "-1";
            theDuration = (smallestDivision / currBarDivision) * restDurationOrig;    //calculate the actual duration taking into account the songs smalles division

            tempBase = new TBaseNote(beatCount, theDuration, chord);    //create the base
            tempDetail = new TNoteDetail(note, theString, theFret);    //create the detail
            tempBase.addNoteDetail(tempDetail);    //add the detail to the base

            allXMLNotes[currLoc] = tempBase;    //add the rest to the array

            beatCount += restDurationOrig;    //increment the time
            currLoc++;
        }
    }
}

```

```

    }
}

}

}

}

//*****

/*This function calculate the smallest division in the entire piece of music*/
public void findSmallestDivision()
{
    smallestDivision = 0;
    int CurrDivision;

    //create a results list of all the smallest division values
    var query = from MusicFile in xml.Elements("score-partwise").Elements("part").Elements("measure").Elements("attributes").Elements("divisions")
                select MusicFile;

    //traverse the results lists and find the smallest
    foreach (var Division in query)
    {
        CurrDivision = Convert.ToInt16(Division.Value);

        if (CurrDivision > smallestDivision)
        {
            smallestDivision = CurrDivision;
        }
    }
}

//*****

/*This function is used to calculate the number of bars within the entire piece of music*/
public void calcNumberOfBars()
{
    int counter = 0;

    //create list containing all measure
    var query = from MusicFile in xml.Elements("score-partwise").Elements("part").Elements("measure")
                select MusicFile;

    //for every measure (bar) increment the counter
    foreach (var Measure in query)
    {
        counter++;
    }

    numberOfBars = counter;

    //debugging
    //MessageBox.Show("Number of bars: " + Convert.ToString(counter));
}

//*****

//This function counts the number of 'notes' in the XML
//It counts each individual note in a chord as 1 note each**
//This function is used for determining the number of notes for the allXMLNotes array (pre chord filtering)
public void calcNumberOfUnsortedNotes()
{
    var query = from MusicFile in xml.Elements("score-partwise").Elements("part").Elements("measure").Elements("note") select MusicFile;

    int counter = 0;
    foreach (var Note in query)
    {
        counter++;
    }

    origNumberOfNotes = counter;

    //debugging
    //MessageBox.Show("Number of notes: " + Convert.ToString(counter));
}

```

```
//*****

/*this function Calculates the number of notes in the piece of music there are.
 * unlike the original song note array that is created at the start and Like the
 * final SongNotes array this counts all notes/elements/parts of a chord as 1 note,
 * also it needs to be noted that rests are counted as a note aswell*/
private void calcNumberOfSortedNotes()
{
    //count all notes withut chord tag...
    var query = from MusicFile in xml.Elements("score-partwise").Elements("part").Elements("measure").Elements("note") select MusicFile;

    int counter = 0;
    foreach (var Note in query)
    {
        /*if doesnt contain chord tage then increment note counter
        (only first note of chord and rests dont contain these, other notes that are
        tagged on the end of previous notes to make a chordinclude the chord tag*/
        if (!Note.Descendants("chord").Any())
            counter++;
    }

    numberOfNotes = counter;

    //debugging
    //MessageBox.Show("Number of notes: " + Convert.ToString(counter));
}
}
```